Cika Electrónica
Nota de Aplicación: CAN-005
Título: **Utilización de displays LCD gráficos (SED1335) con Rabbit 2000**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	16/7/03	

Nos interiorizaremos ahora en el desarrollo de una interfaz para conectar un módulo LCD gráfico inteligente Powertip PG320240FRS, a un módulo Rabbit 2000. Se trata de un display de 320x240 pixels basado en chips controladores compatibles con el SED1335, de S-MOS, y su clon de Epson. Analizaremos más tarde el software de control y un simple programa demostración, que sirve para comprobar el correcto funcionamiento de los módulos LCD que tengamos en stock, y de paso, demostrar sus capacidades. A fin de probar la mayor parte posible del hardware, la interfaz será de 8 bits y realizará lectura y escritura del controlador LCD.

Hardware

El SED1335 presenta una interfaz con dos posibles modos de trabajo: tipo Motorola (E, RS, R/W) o tipo Intel (RD, WR, A0). El PG320240FRS de Powertip lo utiliza en esta última modalidad.
Para la interfaz con el micro no es necesario ningún tipo de glue-logic, hacemos una conexión directa entre los ports del Rabbit y el LCD, al igual que con la gran mayoría de los microcontroladores, como puede apreciarse en la tabla a la derecha:

```
Rabbit        LCD

PA.0  ---------- D0
PA.1  ---------- D1
PA.2  ---------- D2
PA.3  ---------- D3
PA.4  ---------- D4
PA.5  ---------- D5
PA.6  ---------- D6
PA.7  ---------- D7

PE.4  ---------- RD
PE.3  ---------- WR
PE.0  ---------- A0
PE.1  ---------- CS
```

El port A, hace las veces de bus de datos, mientras que los ports libres del port E generarán, por software, las señales de control. La señal CS podría conectarse directamente a masa, a criterio del usuario. El único inconveniente es una posible escritura no intencional al momento del arranque, problema que también podemos tener con este esquema, dado que los ports utilizados son entradas al momento de reset. Podrían incluírse sendos pull-ups si esta posibilidad resultara un inconveniente.

El circuito de contraste de este display es totalmente interno, el mismo puede ajustarse mediante un preset ubicado en la parte posterior del display.
El display dispone, además, de un pin de reset, el cual podemos controlar a voluntad o conectar al reset del circuito. Para el desarrollo de esta nota de aplicación, simplemente lo conectamos mediante un pull-up a la tensión de alimentación.

Software

Breve descripción del display gráfico
Estos displays son sumamente versátiles, la memoria puede ser dividida en diferentes pantallas, las cuales, a su vez pueden definirse como gráficas o de texto, habilitarse y/o superponerse independientemente. Para el caso del modo texto, el SED1335 dispone de un generador de caracteres y una ROM de caracteres de 5x7, aunque es posible utilizar una ROM externa o la misma RAM del display. Al momento de definir cada pantalla, definimos también en qué posición de memoria comienza y cómo se asigna la memoria.
La estructura de memoria de cada pantalla es lineal, tanto en modo gráfico como en modo textol. En este último modo, el primer byte corresponde al caracter ubicado arriba a la izquierda, y el último byte corresponde al ubicado abajo a la derecha. En el modo gráfico, los pixels se agrupan horizontalmente en bytes, correspondiendo el primer byte de memoria a los primeros ocho pixels de la primera línea de arriba a la izquierda, y el último byte a los últimos ocho pixels de la última línea de abajo a la derecha. El bit más significativo del primer byte de memoria corresponde al punto situado en la pantalla arriba a la izquierda, y el bit menos significativo del último byte de memoria corresponde al punto situado en pantalla abajo a la derecha.

El direccionamiento del byte a leer o escribir en memoria se hace mediante comandos, especificando el offset desde la primera dirección correspondiente a la pantalla que nos ocupa. Tiene además un contador autoincrementado, el cual apunta a la dirección siguiente luego de una lectura o escritura. Esto resulta óptimo para enviar los datos byte por byte hasta completar una pantalla.

Una característica interesante del display, es que puede funcionar a una alta velocidad de acceso, simplemente da prioridad a la interfaz con el procesador e interrumpe el display. Si nuestra aplicación requiere que no haya parpadeo (flicker) al momento de escritura, podemos primero chequear el flag de ocupado (busy) y realizar nuestra escritura en el momento que el controlador no accede a la RAM para refrescar el LCD. Si, por el contrario, toleramos el flicker o, como en nuestro caso, hacemos una escritura muy rápida que el mismo no se nota, podemos obviar un paso y escribir directamente sobre el controlador sin chequear el busy flag.

Algoritmos

Si bien el display tiene muchas formas de utilización, en esta nota de aplicación desarrollaremos algoritmos en base a las más simples.

Para direccionar un punto debemos traducir sus coordenadas a una dirección lineal, para ello, deberemos multiplicar la coordenada vertical *y* por la cantidad de bytes en sentido horizontal de la pantalla (40 si utilizamos la totalidad de 320 pixels=40 bytes) y sumarle la coordenada horizontal *x* dividida por 8 (pixels por byte). El resto de dividir *x*/8 es el número de pixel dentro del byte. Dado que el MSB se halla a la izquierda, el pixel 0 correponde al bit 7 y el pixel 7 al bit 0, es decir: *address=40*y+x/8 ; bit=7-resto(x/8).*

Para graficar funciones, debemos tener en cuenta que la coordenada (0;0) se halla en el extremo superior izquierdo de la pantalla.

Para mostrar pantallas, deberemos agrupar los datos de modo tal de poder enviarlos de forma que aproveche de manera eficiente los contadores autoincrementados y la estructura de memoria; dada la estructura lineal, esto se reduce simplemente a enviar todos los bytes corridos. Si comparamos la estructura de memoria del display con la forma de guardar imágenes blanco y negro en formato BMP, veríamos que son muy similares, por ejemplo: BMP va de abajo a arriba y el display de arriba a abajo, por lo que la imagen se ve espejada verticalmente; BMP usa el '0' para el negro y el '1' para el blanco, si el display es STN+, '1' corresponde a un punto negro. Además, BMP incluye un encabezado de 62 bytes.

Por consiguiente, para adaptar una imagen, debemos llevarla a la resolución deseada, espejarla verticalmente, invertir sus colores, salvarla en formato BMP y por último descartar los 62 bytes del comienzo con algún editor hexa.

Para imprimir textos, calculamos simplemente la posición de memoria a partir de fila y columna de modo similar: *address=40*fila+columna*, para 40 caracteres por fila (matriz de caracteres de 8x8).

Desarrollo

Desarrollamos a continuación el software de base para manejo del display. Definiremos dos pantallas: una gráfica de 320x240 y una de texto de 40x30, con caracteres de 8x8, usando el generador interno, por lo que se verán caracteres de 5x7 en una trama de 8x8. Para una mejor comprensión, dada la complejidad del SED1335, se recomienda consultar su manual técnico.

Dada la gran cantidad de información a mover hacia el display para la presentación de pantallas, decidimos escribir la rutina básica de escritura en assembler. Dado que solamente nuestra rutina accede al port A, prescindimos de su correspondiente shadow register; no obstante, dado que el port E puede compartirse con otra tarea, creemos “buena práctica” el actualizar el shadow register. Esta actualización debe hacerse de forma atómica, para evitar que una interrupción altere el valor en la mitad de la operación.

```
/* LCD control signals */
#define LCD_RD  4
#define LCD_A0  0
#define LCD_WR  3
#define LCD_CS  1

/* Low level functions */

#asm
;function requires one parameter:
;@sp+2= data to write
;
LCD_Write::
```

```
        ld hl,(sp+2)                ; extrae del stack el valor a escribir
        ld a,l                      ; usa solo el LSB (unsigned char = byte)
        ioi ld (PADR),a             ; lo escribe, PA0-7 se mantienen como salidas
        ld hl,PEDRShadow            ; apunta al control del port E (shadow)
        ld de,PEDR                  ; apunta al control del port E
        res LCD_WR,(HL)             ; Baja WR
        ioi ldd                     ; ahora (atomic)
        ld hl,PEDRShadow            ; apunta al control del port E (shadow)
        ld de,PEDR                  ; apunta al control del port E
        set LCD_WR,(HL)             ; Sube WR
        ioi ldd                     ; ahora
        ret

#endasm
```

El prefijo *ioi* es el que nos permite utilizar cualquier instrucción de acceso a memoria como instrucción de I/O. Esta es una de las mayores diferencias entre Rabbit y Z-80, en cuanto a set de instrucciones; descartando, claro está las funciones agregadas. Obsérvese como la operación atómica se realiza mediante la instrucción ldd (LoaD and Decrement), que copia hacia la posición apuntada por DE el contenido de la posición apuntada por HL, decrementando ambos punteros y el contador BC. El inconveniente es que más adelante debemos volver a cargar el mismo valor en los punteros (preferimos esto en vez de salvarlos en el stack porque es más rápido), o incrementarlos nuevamente (más rápido aún). Nos pareció menos confuso para el desarrollo de la nota volver a cargar el mismo valor, sacrificando eficiencia por claridad.

El resto de las funciones se ha escrito en C, dado que con el incremento de velocidad logrado es suficente para el módulo utilizado y las prestaciones esperadas de una nota de aplicación.

```
void LCD_WriteCmd(unsigned char cmd)
{
        BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_A0 );  // Sube A0 (Cmd)
        LCD_Write(cmd);
        BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_A0 );  // Baja A0 (Data)
}

void LCD_WriteStrCmd(unsigned char *cmd,int len)
{
        LCD_WriteCmd(*cmd++);                        // manda comando
        while(len--) {                               // manda parámetros
                LCD_Write(*cmd++);
        }
}

unsigned char LCD_ReadData()
{
unsigned char data;
        WrPortI ( SPCR, &SPCRShadow, 0x80 );         // PA0-7 = Inputs
        BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_A0 );  // Sube A0 (Data)
        BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_RD );  // Baja RD
        data=RdPortI(PADR);                          // lee datos del bus
        BitWrPortI ( PEDR, &PEDRShadow, 1,LCD_RD );  // Sube RD
        BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_A0 );  // Baja A0 (Sts)
        WrPortI ( SPCR, &SPCRShadow, 0x84 );         // PA0-7 = Outputs
        return(data);
}
```

Algunas funciones de soporte, para generar demoras

```
void MsDelay ( int iDelay )
{
        unsigned long ul0;
        ul0 = MS_TIMER;                              // valor actual del timer
        while ( MS_TIMER < ul0 + (unsigned long) iDelay );
}
```

Veamos ahora la función de inicialización, la cual es algo extensa dadas las prestaciones del SED1335 y su inherente complejidad.

```
void LCD_init ()
{
const static unsigned char init_string1[]={
0x40,                  // INIT
'\B00110000',          // 32 char CGRAM, 8 pixel chars, no top-screen corr, LCD, normal
0x87,0x7,              // 8x8 characters
39,59,                 // 320 pixels, 8 pixel characters => 40 characters per line
239,                   // 240 lines (30 txt lines)
40,0                   // virtual screen = display screen
};

const static unsigned char init_string2[]={
0x44,                  // SCROLL
0,                     // SAD1L: screen 1 empieza en (0x0000)
0,                     // SAD1H:
239,                   // SL1  : scrolling lines (239 dec.)
0xB0,                  // SAD2L: screen 2 empieza en (1200=0x4b0)
0x04,                  // SAD2H:
239,                   // SL2  : scrolling lines
0,                     // SAD3L:
0,                     // SAD3H:
0,                     // SAD4L:
0                      // SAD4H:
};

const static unsigned char init_string3[]={
0x5D,                  // CSRFORM
0x04,                  // CRX: cursor horiz (4 pixel)
0x86                   // CRY: cursor vert (6 pixel) CM : block cursor
};


const static unsigned char init_string4[]={
0x5A,                  // HDOT_SCR (0x5a), scroll rate.
0                      // 1 pixel scroll
};

const static unsigned char init_string5[]={
0x5B,                  // OVLAY
'\B00000001'           // XOR screens 1 y 2
};

const static unsigned char init_string6[]={
0x59,                  // DISP_ON
'\B00010110'           // cursor flash = 2 Hz, SAD1/2 no flashing, SAD3 off
};

        WrPortI ( PEDR,&PEDRShadow,'\B00011010' );          // CS,RD,WR = HIGH
        WrPortI ( PEDDR,&PEDDRShadow,'\B10011011' );        // PE0,1,4,3,7 = output
        WrPortI ( SPCR, &SPCRShadow, 0x84 );                // PA0-7 = Outputs
        MsDelay ( 1000 );                                   // espera LCD reset

        BitWrPortI ( PEDR, &PEDRShadow, 0,LCD_CS );         // Baja CS

        LCD_WriteStrCmd ( init_string1,sizeof(init_string1) ); // inicializa el display
        LCD_WriteStrCmd ( init_string2,sizeof(init_string2) );
        LCD_WriteStrCmd ( init_string3,sizeof(init_string3) );
        LCD_WriteCmd ( 0x4C );        // CSRDIR_RIGHT, cursor incrementa a la derecha
        LCD_WriteStrCmd ( init_string4,sizeof(init_string4) );
        LCD_WriteStrCmd ( init_string5,sizeof(init_string5) );
        LCD_WriteStrCmd ( init_string6,sizeof(init_string6) );
```

```
}
```

Rutinas de soporte de alto nivel, sumamente simples

```
/* High level functions */

void LCD_cursor ( int address )
{
	LCD_WriteCmd(0x46);				// CSRW.
	LCD_Write(address&0xFF);			// LO byte
	LCD_Write((address>>8)&0xFF);			// HI byte
}

void LCD_fill(unsigned char pattern, int len)
{
int i;
	LCD_WriteCmd(0x42);				// MWRITE
	while(len--)
		LCD_Write(pattern);			// llena con pattern
}

void LCD_cleargfx ( void )
{
	LCD_cursor(1200);		// direcciona la pantalla gráfica
	LCD_fill(0,9600);		// borra pantalla (llena con 0's)
}

void LCD_cleartxt ( void )
{
	LCD_cursor(0);			// direcciona pantalla de texto
	LCD_fill(' ',1200);		// borra pantalla (llena con espacios)
}

void LCD_printat (unsigned int row, unsigned int col, char *ptr)
{
	LCD_cursor (40*row+col);	// posiciona cursor
	LCD_WriteCmd(0x42);		// MWRITE
	while (*ptr)			// escribe caracteres
		LCD_Write (*ptr++);
}

/* Plot image data, 8 pixels per byte, MSB left */

void LCD_plot (int x, int y)
{
int strip,bit,data;
	strip=x>>3;			// strip = x/8
	bit=x&'\B0111';			// bit = resto
	y=1200+strip+40*y;		// pantalla gráfica empieza en address 1200
	LCD_cursor(y);			// direcciona strip en pantalla gráfica
	LCD_WriteCmd(0x43);		// MREAD
	data=LCD_ReadData();		// lee
	data|=(0x80>>bit);		// set bit (MSB a la izquierda)
	LCD_cursor(y);			// vuelve a direccionar (autoincrementa)
	LCD_WriteCmd(0x42);		// MWRITE
	LCD_Write(data);		// escribe
}
```

Para mostrar pantallas, dado que la extensión de las mismas es de 9600 bytes, puede no ser conveniente cargarlas en el espacio base, por lo que nos conviene usar el espacio extendido de memoria. Afortunadamente, Dynamic C provee una función que permite, al momento de compilación, leer un archivo y ubicarlo para

ocupar un área de memoria, asociando esa posición de memoria física con un puntero extendido que hace referencia a esa posición. El área apuntada corresponde a una estructura del tipo:

puntero (long) -> longitud de los datos ('n', long)
datos propiamente dichos (n bytes)

De esta forma, resulta muy fácil incluir las imágenes al programa:

```
#ximport "cika.pbm" cika
#ximport "rabtip.pbm" rabtip
#ximport "goku.pbm" goku
```

El único inconveniente de trabajar de esta forma, es que acceder a la memoria por su dirección física resulta algo engorroso, por lo que es preferible copiarla al área base y transferirla de allí al display. Dynamic C trae funciones para hacer ésto. Para utilizar poca memoria, partimos la imagen en bloques, movemos estos bloques de una a otra área y copiamos al display. En este caso, serían 40 bloques de 240 bytes c/u.

```
#define IMGCHUNK 240
#define CHUNKS 40

void LCD_dump(long imgdata)
{
int x,y;
unsigned char buffer[IMGCHUNK];

        LCD_cursor(1200);                    // selecciona pantalla gráfica
        LCD_WriteCmd(0x42);                  // MWRITE
        for(y=0;y<CHUNKS;y++){
          xmem2root(&buffer,imgdata+IMGCHUNK*y+sizeof(long),IMGCHUNK); // trae bloque de xmem
          for(x=0;x<IMGCHUNK;x++)
                LCD_Write(buffer[x]);        // copia al display
        }
}
```

Mostraremos ahora unas simples funciones para hacer gráficos de tres dimensiones en perspectiva. Las mismas tienen como objeto demostrar una aplicación posible del display y han sido portadas de viejos programas BASIC, por lo que son mucho menos que óptimas. No obstante, puede observarse que trabajar con números en coma flotante es sumamente fácil, y podrá además apreciarse la velocidad de cálculo de la combinación Rabbit-Dynamic C, ya que la función se dibuja mientras se va calculando.

```
/* simple 3D function plotting */

void paraboloide(int PERSP,float SCALE,int TRAZOSX,int TRAZOSY,float X0,float X1,float
Y0,float Y1)
{
auto int xx,yy;
auto float x,y,z;
auto float persp,xt,zt;
auto float h1,h2,h3,h4,cs,si;

        persp=PERSP*0.01745329;                  // convierte a radianes
        cs=cos(persp);                           // calcula perspectiva
        si=sin(persp);
        h1=(X1-X0)/159;                          // calcula paso de barrida
        h2=(Y1-Y0)/(float)(TRAZOSX-1);
        h3=(X1-X0)/(float)(TRAZOSY-1);
        h4=(Y1-Y0)/159;

        for(y=Y0;y<=Y1;y+=h2)                    // barre en un sentido
                for(x=X0;x<=X1;x+=h1){
                        z=x*x/4+y*y/9-15;
                        xt=x-y*cs;
                        zt=z-y*si;
                        xx=160+(int)(SCALE*xt);
                        yy=120-(int)(SCALE*zt);
```

```
                        LCD_plot(xx,yy);
                }                                    // barre en el otro sentido
        for(x=X0;x<=X1;x+=h3)
                for(y=Y0;y<=Y1;y+=h4) {
                        z=x*x/4+y*y/9-15;
                        xt=x-y*cs;
                        zt=z-y*si;
                        xx=160+(int)(SCALE*xt);
                        yy=120-(int)(SCALE*zt);
                        LCD_plot(xx,yy);
                }
}

void elipsoide(int PERSP,float SCALE,int TRAZOSX,int TRAZOSY,float X0,float X1,float Y0,float
Y1)
{
<eliminada por cuestiones de espacio, muy similar a paraboloide>
}
```

El siguiente fragmento de código es un simple ejemplo de la utilización de Dynamic C para escribir un corto y simple programa de control que nos permita demostrar el funcionamiento del display LCD. Para observar si todos los pixels funcionan correctamente, pintamos la pantalla de negro y luego la blanqueamos. Luego graficamos unas funciones en 3D y finalmente mostramos unas imágenes.

```
/* MAIN PROGRAM */

main()
{
int i;

        LCD_init();
        while(1){

                LCD_cleartxt();                         // borra pantalla de texto
                LCD_cleargfx();                         // borra pantalla gráfica
                MsDelay ( 1000 );                       // espera 1 seg
                LCD_cursor(1200);                       // pantalla gráfica
                LCD_fill(0xFF,9600);                    // pone en negro
                MsDelay ( 3000 );                       // espera 3 segs
                LCD_cleargfx();                         // borra
                LCD_printat(0,0,"Graficos en 3D en tiempo real:"); // texto superpuesto
                paraboloide(30,6,20,20,-8.0,8.0,-10.0,10.0); // gráfico 3D
                MsDelay ( 10000 );                      // espera 10 segs
                LCD_cleartxt();                         // borra texto anterior
                LCD_dump(cika);                         // muestra logo Cika
                MsDelay ( 3000 );                       // espera 3 segs
                LCD_cleargfx();                         // borra logo
                LCD_printat(0,0,"Graficos en 3D en tiempo real:");
                elipsoide(45,80.0,30,30,0,6.28,-3.14,3.14);  // gráfico 3D
                MsDelay ( 10000 );                      // espera 10 segs
                LCD_cleartxt();                         // borra texto anterior
                LCD_dump(goku);                         // muestra imagen
                LCD_printat(0,0,"Texto y graficos");    // superpone texto
                LCD_printat(1,2,"superpuestos");
                MsDelay ( 6000 );                       // espera 3 segs
                LCD_cleartxt();                         // borra texto
                LCD_dump(rabtip);                       // muestra logos
                LCD_printat(25,8,"Cika Electronica S.R.L."); // agrega texto
                LCD_printat(29,30,"julio 2003");
                MsDelay ( 10000 );                      // espera 10 segs
        }
}
```